

UNITED STATES PATENT APPLICATION FOR

CAPTURING TEST/EMULATION AND ENABLING REAL-TIME DEBUGGING
USING AN FPGA FOR IN-CIRCUIT EMULATION

Inventors:

Warren Snyder

Prepared by:
WAGNER, MURABITO & HAO, LLP
Two North Market Street
Third Floor
San Jose, California 95113
(408) 938-9060

1
2
3
4
5
6 **CAPTURING TEST/EMULATION AND ENABLING REAL-TIME**
7 **DEBUGGING USING AN FPGA FOR IN-CIRCUIT EMULATION**
8
9
10

11 **CROSS REFERENCE TO RELATED DOCUMENTS**

12 This application is related to U.S. Patent Application Serial No.
13 _____, docket number CYPR-CD00182, to Warren Snyder, et al., entitled
14 "IN-SYSTEM CHIP EMULATOR ARCHITECTURE"; and to U.S. Patent Application
15 Serial No. _____, docket number CYPR-CD00185, to Warren Snyder, et al.,
16 entitled "EMULATOR CHIP/BOARD ARCHITECTURE AND INTERFACE"; and to
17 U.S. Patent Application Serial No. _____, docket number CYPR-CD00182,
18 to Warren Snyder, et al., entitled "METHOD FOR BREAKING EXECUTION OF
19 (TEST) CODE IN A DUT AND EMULATOR CHIP ESSENTIALLY
20 SIMULTANEOUSLY AND HANDLING COMPLEX BREAKPOINT EVENTS"; and to
21 U.S. Patent Application Serial No. _____, docket number CYPR-CD00184,
22 to Craig Nemecek, entitled "HOST TO FPGA INTERFACE IN AN IN-CIRCUIT
23 EMULATION SYSTEM. Each of these applications is filed on the same date as the
24 present application and is hereby incorporated by reference as though disclosed
25 fully herein. This application is also related to and claims priority benefit under 35
26 U.S.C. §119(e) of provisional patent application serial no. 60/243,708 filed October
27 26, 2000 to Snyder, et al. entitled "Advanced Programmable Microcontroller Device"
28 which is hereby incorporated herein by reference.
29

FIELD OF THE INVENTION

This invention relates generally to the field of in-circuit emulation. More particularly, this invention relates to a method for obtaining real-time debugging information from an FPGA operating as a "virtual" device coupled to the device under test.

BACKGROUND OF THE INVENTION

In-circuit emulation (ICE) has been used by software and hardware developers for a number of years as a development tool to emulate the operation of complex circuit building blocks and permit diagnosis and debugging of hardware and software. Such in-circuit emulation is most commonly used currently to analyze and debug the behavior of complex devices such as microcontrollers and microprocessors that have internal structures that are far too complex to readily model using computer simulation software alone.

FIGURE 1 illustrates an exemplary conventional in-circuit emulation arrangement 100 used to model, analyze and debug the operation of a microcontroller device. In this arrangement, a host computer (e.g., a personal computer) 110 is connected to a debug logic block 120 which is further connected to a special version of the microcontroller device that has been developed specially for use in emulation. Operational instructions are loaded from the host computer 110 through the debug logic 120 to the special version of the microcontroller 130. The debug logic 120 monitors operation of the microcontroller 130 as the instructions are executed. Depending upon the application, this operation may be monitored while the special version of the microcontroller 130 is interconnected with the circuitry that is intended to interface a production version of the microcontroller in the finished product under development. Such interconnection may be via simulation within host computer 110 or as actual circuitry or some combination thereof. As the circuit is stepped through its operation, the debug logic gathers information about the state of various components of the

1 microcontroller 130 during operation and feeds that information back to the host
2 computer 110 for analysis.

3 During the course of the analysis, various trace information such as time
4 stamps, register values, data memory content, etc. may be logged in the host
5 computer 110 for analysis and debugging by the designer. Additionally, it is
6 generally the case that various break points can be defined by the designer that
7 cause the program to halt execution at various points in the operation to permit
8 detailed analysis. Other debugging tools may also be provided to enable the user
9 to debug the operation of the circuit.

10 In-circuit emulation systems such as 100 have a number of disadvantages
11 and limitations. In earlier systems, the microcontroller 130 might have been simply
12 the production version of the microcontroller itself with no special debugging
13 features. In such systems, the information that can be gathered by the ICE system
14 100 is limited to that which is available at the pinouts of the microcontroller 130 (or
15 which can be extracted from the microcontroller using clever programming or
16 special coding supported by the processor).

17 Enhancements to such early systems provided the microcontroller or other
18 processor with an array of built-in debugging tools that use standard pins on the
19 part and built-in instructions that facilitated in-circuit emulation. In such enhanced
20 processors, the emulation tools are integrated into the part and thus become a
21 design constraint for developing and improving the part. Thus, support for the
22 debugging instruction code and the like can increase the cost and complexity of the
23 circuit.

24 Newer systems often use a so-called "bond-out" microcontroller. A bond-out
25 version of the microcontroller is a version of the production microcontroller that has
26 been designed with special wirebonding pads on the chip that are not normally
27 connected in the production wirebonding. The bond-out version connects these
28 pads to pins on the microcontroller package to permit access to otherwise
29 inaccessible points of the circuit to facilitate debugging. This technique is in
30 common use, but has the disadvantage of imposing significant limitations on the

1 circuit layout to permit space and circuitry associated with the special wirebonding
2 pads. Additionally, it is usually the case that substantial interface circuitry and
3 other special circuitry to facilitate the debugging and bond-out has to be added to
4 the circuit. This increases the complexity, size, power consumption and potentially
5 reduces the yield of the production part. Moreover, development resources are
6 required to lay out the bond-out circuitry and pads and do associated design of
7 such bond-out circuitry. Additionally, instruction code must generally be provided
8 and supported for such an implementation. Such resources may have to be
9 applied with every updated version of the part and may significantly impact speed,
10 cost or flexibility in development of improved versions of the part.

11 A third technique, one that is used in the Pentium™ and Pentium Pro™
12 series of microprocessors available from Intel Corporation, provides a special
13 "probe mode" of operation of the processor. When the processor is placed in this
14 probe mode, a number of internal signals are routed to a "debug port" for use by the
15 in-circuit emulation system. This debug port is used to permit the in-circuit
16 emulation system to communicate with the processors at all times and, when
17 placed in probe mode, to read otherwise inaccessible probe points within the
18 processor. Of course, providing such a probe mode requires significant design
19 resources to design in all such probe and debug functions and associated
20 instruction code support into the standard processor. This, of course, increases
21 development cost, chip complexity and chip size. Moreover, such facilities become
22 a part of the processor design which must be carried through and updated as
23 required as enhancements to the original design are developed.

24 25 **SUMMARY OF THE INVENTION**

26 The present invention relates generally to in-circuit emulation. Objects,
27 advantages and features of the invention will become apparent to those skilled in
28 the art upon consideration of the following detailed description of the invention.

29 In one embodiment consistent with the present invention, a method is
30 provided for obtaining real-time debug information, e.g., state information and trace

1 information, from an FPGA acting as a virtual microcontroller that is attached to a
2 microcontroller under test. The two devices, the microcontroller and the FPGA
3 execute the same instructions in lock-step with the FPGA acting as an emulator.
4 The FPGA emulates the actual microcontroller and relieves the actual
5 microcontroller from having debug logic installed thereon. FPGA and
6 microcontroller, are coupled using a four pin interface. The FPGA is directly
7 coupled to the PC for both programming and control. The system is implemented
8 such that the microcontroller forwards information regarding I/O reads, interrupt
9 vector information and watchdog information to the FPGA in time before the
10 execution of the next instruction. Thus, the FPGA has an exact copy of the state
11 information of the microcontroller. This method has many advantages such as for
12 example, robust debugging capability required for complex programmable
13 microcontroller chip, delivery of real-time trace and state information to a logic
14 analyzer system without interrupting the device under test; and the FPGA
15 eliminates the need for designing and maintaining a special microcontroller chip
16 specially adapted just for debugging, this accomplished by providing a "virtual
17 microcontroller" using the FPGA.

18 A method of obtaining debug information, in an embodiment consistent with
19 the present invention includes executing a sequence of instructions by a device
20 under test (DUT); executing the sequence of instructions by an emulator device
21 emulating the functions of the DUT and executing the sequence of instructions in
22 lock-step fashion with the DUT; the DUT conveying I/O read information to the
23 emulator device; and a host computer system reading real-time state and debug
24 information from the emulator device without interrupting the DUT.

25 Another method of obtaining debug information, consistent with an
26 embodiment of the present invention includes a) executing a sequence of
27 instructions by a microcontroller device; b) in synchronization with a), an emulator
28 device emulating the functions of the microcontroller and executing the sequence
29 of instructions in lock-step fashion with the microcontroller; c) the microcontroller
30 conveying I/O read information to the emulator device; and d) a host computer

1 system reading real-time state and debug information from the emulator without
2 interrupting the microcontroller.

3 The above summaries are intended to illustrate exemplary embodiments of
4 the invention, which will be best understood in conjunction with the detailed
5 description to follow, and are not intended to limit the scope of the appended
6 claims.

7 8 **BRIEF DESCRIPTION OF THE DRAWINGS**

9 The features of the invention believed to be novel are set forth with
10 particularity in the appended claims. The invention itself however, both as to
11 organization and method of operation, together with objects and advantages
12 thereof, may be best understood by reference to the following detailed description
13 of the invention, which describes certain exemplary embodiments of the invention,
14 taken in conjunction with the accompanying drawings in which:

15 **FIGURE 1** is a block diagram of a conventional In-Circuit Emulation system.

16 **FIGURE 2** is a block diagram of an exemplary In-Circuit Emulation system
17 consistent with certain microcontroller embodiments of the present invention.

18 **FIGURE 3** is an illustration of the operational phases of an In-Circuit
19 Emulation system consistent with an embodiment of the present invention.

20 **FIGURE 4** is an illustration of the operational phases of an In-Circuit
21 Emulation system consistent with an embodiment of the present invention viewed
22 from a virtual microcontroller perspective.

23 **FIGURE 5** is a timing diagram useful in understanding an exemplary data
24 and control phase of operation of certain embodiments of the present invention.

25 26 **DETAILED DESCRIPTION OF THE INVENTION**

27 In the following detailed description of the present invention, numerous
28 specific details are set forth in order to provide a thorough understanding of the
29 present invention. However, it will be recognized by one skilled in the art that the

1 present invention may be practiced without these specific details or with
2 equivalents thereof. In other instances, well known methods, procedures,
3 components, and circuits have not been described in detail as not to unnecessarily
4 obscure aspects of the present invention.
5

6 NOTATION AND NOMENCLATURE

7 Some portions of the detailed descriptions which follow are presented in
8 terms of procedures, steps, logic blocks, processing, and other symbolic
9 representations of operations on data bits that can be performed on computer
10 memory. These descriptions and representations are the means used by those
11 skilled in the data processing arts to most effectively convey the substance of their
12 work to others skilled in the art. A procedure, computer executed step, logic block,
13 process, etc., is here, and generally, conceived to be a self-consistent sequence
14 of steps or instructions leading to a desired result. The steps are those requiring
15 physical manipulations of physical quantities.

16 Usually, though not necessarily, these quantities take the form of electrical
17 or magnetic signals capable of being stored, transferred, combined, compared, and
18 otherwise manipulated in a computer system. It has proven convenient at times,
19 principally for reasons of common usage, to refer to these signals as bits, values,
20 elements, symbols, characters, terms, numbers, or the like.

21 It should be borne in mind, however, that all of these and similar terms are
22 to be associated with the appropriate physical quantities and are merely convenient
23 labels applied to these quantities. Unless specifically stated otherwise as apparent
24 from the following discussions, it is appreciated that throughout the present
25 invention, discussions utilizing terms such as "processing" or "transferring" or
26 "executing" or "determining" or "instructing" or "issuing" or "halting" or the like, refer
27 to the action and processes of a computer system, or similar electronic computing
28 device, that manipulates and transforms data represented as physical (electronic)
29 quantities within the computer system's registers and memories into other data
30 similarly represented as physical quantities within the computer system memories

or registers or other such information storage, transmission or display devices.

CAPTURING TEST/EMULATION AND ENABLING REAL-TIME DEBUGGING USING AN FPGA FOR IN-CIRCUIT EMULATION IN ACCORDANCE WITH THE INVENTION

While this invention is susceptible of embodiment in many different forms, there is shown in the drawings and will herein be described in detail specific embodiments, with the understanding that the present disclosure is to be considered as an example of the principles of the invention and not intended to limit the invention to the specific embodiments shown and described. In the description below, like reference numerals are used to describe the same, similar or corresponding parts in the several views of the drawings.

A commercial ICE system utilizing the present invention is available from Cypress Micro Systems, Inc., for the CY8C25xxx/26xxx series of microcontrollers. Detailed information regarding this commercial product is available from Cypress Micro Systems, Inc., 22027 17th Avenue SE, Suite 201, Bothell, WA 98021 Bothell, WA in the form of version 1.11 of "PSoC Designer: Integrated Development Environment User Guide", which is hereby incorporated by reference. While the present invention is described in terms of an ICE system for the above exemplary microcontroller device, the invention is equally applicable to other complex circuitry including microprocessors and other circuitry that is suitable for analysis and debugging using in-circuit emulation. Moreover, the invention is not limited to the exact implementation details of the exemplary embodiment used herein for illustrative purposes.

Referring now to **FIGURE 2**, an architecture for implementation of an embodiment of an ICE system of the present invention is illustrated as system 200. In system 200, a Host computer 210 (e.g., a personal computer based on a Pentium™ class microprocessor) is interconnected (e.g., using a standard PC interface 214 such as a parallel printer port connection, a universal serial port

1 (USB) connection, etc.) with a base station 218. The host computer 210 generally
2 operates to run an ICE computer program to control the emulation process and
3 further operates in the capacity of a logic analyzer to permit a user to view
4 information provided from the base station 218 for use in analyzing and debugging
5 a system under test or development.

6 The base station 218 is based upon a general purpose programmable
7 hardware device such as a gate array configured to function as a functionally
8 equivalent "virtual microcontroller" 220 (or other device under test (DUT)). This is
9 accomplished using an associated integral memory 222 which stores program
10 instructions, data, trace information and other associated information. Thus, the
11 base station is configured as an emulator of the internal microprocessor portion of
12 the microcontroller 232. In preferred embodiments, a field programmable gate
13 array FPGA (or other programmable logic device) is configured to function as the
14 virtual microcontroller 220. The FPGA and virtual microcontroller 220 will be
15 referred to interchangeably herein. The base station 218 is coupled (e.g., using a
16 four wire interface 226) to a standard production microcontroller 232 mounted in a
17 mounting device referred to as a "pod". The pod, in certain embodiments, provides
18 connections to the microcontroller 232 that permit external probing as well as
19 interconnection with other circuitry as might be used to simulate a system under
20 development.

21 The FPGA of the base station 218 of the current embodiment is designed
22 to emulate the core processor functionality (microprocessor functions, Arithmetic
23 Logic Unit functions and RAM and ROM memory functions) of the Cypress
24 CY8C25xxx/26xxx series microcontrollers. The CY8C25xxx/26xxx series of
25 microcontrollers also incorporates I/O functions and an interrupt controller as well
26 as programmable digital and analog circuitry. This circuitry need not be modeled
27 using the FPGA 220. Instead, the I/O read information, interrupt vectors and other
28 information can be passed to the FPGA 220 from the microcontroller 232 over the
29 interface 226 as will be described later.

30 In order to minimize the need for any special ICE related functions on the

1 microcontroller 232 itself, the FPGA 220 and associated circuitry of the base station
2 218 are designed to operate functionally in a manner identically to that of
3 microprocessor portion of the production microcontroller, but to provide for access
4 to extensive debug tools including readout of registers and memory locations to
5 facilitate traces and other debugging operations.

6 The base station 218's virtual microcontroller 220 operates to execute the
7 code programmed into the microcontroller 232 in lock-step operation with the
8 microcontroller 232. Thus, the actual microcontroller 232 is freed of any need to
9 provide significant special facilities for ICE, since any such facilities can be
10 provided in the virtual microcontroller 220. The base station 218's virtual
11 microcontroller 220 and microcontroller 232 operate together such that I/O reads
12 and interrupts are fully supported in real time. The combination of real and virtual
13 microcontroller behave just as the microcontroller 232 would alone under normal
14 operating conditions. I/O reads and interrupt vectors are transferred from the
15 microcontroller 232 to the base station 218 as will be described later. Base station
16 218 is then able to provide the host computer 210 with the I/O reads and interrupt
17 vectors as well as an array of information internal to the microcontroller 232 within
18 memory and register locations that are otherwise inaccessible.

19 In the designing of a microcontroller other complex circuit such as the
20 microcontroller 232, it is common to implement the design using the Verilog™
21 language (or other suitable language). Thus, it is common that the full functional
22 design description of the microcontroller is fully available in a software format. The
23 base station 218 of the current embodiment is based upon the commercially
24 available Spartan™ series of FPGAs from Xilinx, Inc., 2100 Logic Drive, San Jose,
25 CA 95124. The Verilog™ description can be used as the input to the FPGA design
26 and synthesis tools available from the FPGA manufacturer to realize the virtual
27 microcontroller 220 (generally after timing adjustments and other debugging).
28 Thus, design and realization of the FPGA implementation of an emulator for the
29 microcontroller (virtual microcontroller) or other device can be readily achieved by
30 use of the Verilog™ description along with circuitry to provide interfacing to the base

1 station and the device under test (DUT).

2 In the embodiment described in connection with **FIGURE 2**, the actual
3 production microcontroller 232 carries out its normal functions in the intended
4 application and passes I/O information and other information needed for debugging
5 to the FPGA 220. The virtual microcontroller 220 implemented within the FPGA of
6 base station 218 serves to provide the operator with visibility into the core processor
7 functions that are inaccessible in the production microcontroller 232. Thus, the
8 FPGA 220, by virtue of operating in lock-step operation with the microcontroller 232
9 provides an exact duplicate of internal registers, memory contents, interrupt vectors
10 and other useful debug information. Additionally, memory 222 can be used to store
11 information useful in trace operations that is gathered by the FPGA 220 during
12 execution of the program under test. This architecture, therefore, permits the
13 operator to have visibility into the inner workings of the microcontroller 232 without
14 need to provide special bondouts and expensive circuitry on the microcontroller
15 itself.

16 The base station 218's FPGA based virtual microcontroller 220, operating
17 under control of host computer 210, carries out the core processor functions of
18 microcontroller 232 and thus contains a functionally exact emulated copy of the
19 contents of the registers and memory of the real microcontroller 232. The ICE
20 system starts both microcontrollers (real and virtual) at the same time and keeps
21 them running in synchronization. The real microcontroller 232 sends I/O data to
22 the base station 218 (and in turn to the ICE software operating on the host
23 computer 210 if required) fast enough to keep the real microcontroller 232 and the
24 virtual microcontroller 220 of base station 218 in synchronization. Whenever the
25 system is halted (i.e., when the system is not emulating), other information such
26 as flash memory programming functions, test functions, etc. can be sent over the
27 interface.

28 Because the microcontroller 232 operates in synchronization with the virtual
29 microcontroller 220, less data needs to be sent over the four wire interface than
30 would be required in an ICE system otherwise. The type of data sent over the lines

1 is allowed to change depending on when the data is sent in the execution
2 sequence. In other words, depending on the execution sequence time, the
3 information over the data lines can be commands to the real microcontroller 232
4 or they can be data. Since the clock frequency of the real microcontroller 232 is
5 programmable, it copies its current clock on one of the lines of the four wire
6 interface. Moreover, the lock-step operation of the microcontroller 232 and the
7 virtual microcontroller 220 allows the virtual microcontroller 220 to not require
8 certain resources of the microcontroller 232 such as timers, counters, amplifiers,
9 etc. since they are fully implemented in the microcontroller 232. In addition, the
10 microcontroller 232 (or other DUT) can be debugged in real time without need for
11 extensive debug logic residing on the microcontroller 232, since all registers and
12 memory locations, etc. are available through the virtual microcontroller 220.

13 In the embodiment illustrated, the basic interface used is a four line interface
14 226 between microcontroller 232 and base station 218. This interface permits use
15 of a standard five wire Category Five patch cable to connect the microcontroller 232
16 and base station 218 in one embodiment, but of course, this is not to be considered
17 limiting. The four wire interface 226 of the present embodiment can be functionally
18 divided into two functional portions. A data transport portion 242 carries two data
19 lines in the current embodiment. A clock portion 246 carries a data clock signal
20 plus the microcontroller clock signal for the microcontroller 232. Three additional
21 lines are also provided (not shown) for supply, ground and a reset line. But, the
22 data transport portion 242 and the clock portion 246 are of primary interest, since
23 the supply and reset functions can be readily provided in any other suitable manner.

24 The two portions of the interface are implemented in the current embodiment
25 using four lines as described, however, in other embodiments, these two portions
26 can be implemented with as few as two wires. In the current embodiment, the
27 microcontroller clock signal can be varied by programming (even dynamically
28 during execution of a program). Therefore, it is desirable to have two clock signals
29 - the microcontroller clock to easily track the microcontroller clock timing as well
30 as a system clock that regulates the data transfer and other operations. However,

1 in other embodiments, particularly where a clock frequency is not changed
2 dynamically, a single clock can be used. The single clock can be multiplied or
3 divided as required to implement the required clocking signals.

4 The present embodiment using an eight bit microcontroller that only reads
5 eight bits at a time on any given I/O read. Thus, the present microcontroller 232
6 needs only to effect serializing and transferring a maximum of one eight bit I/O read
7 for each instruction cycle. This is easily accommodated using two data lines
8 transferring four bits each over four system clock cycles. However, using a clock
9 which is two times faster, a single line could equally well transfer the data in the
10 same time. Similarly, four lines could be used to transfer the same data in only two
11 clock cycles. In any case, the objective is to transfer the data in a short enough
12 time to permit the virtual microcontroller 220 to process the data and issue any
13 needed response before the next instruction cycle begins. The time required to
14 accomplish this is held at a minimum in the current invention, since the system
15 synchronization eliminates need for any overhead protocol for transmission of the
16 data.

17 The current embodiment of the invention uses a four line communication
18 interface and method of communicating between the FPGA within base station 218
19 (acting as a "virtual microcontroller" 220 or ICE) and the real microcontroller device
20 under test (microcontroller 232). The four line communication interface is time-
21 dependent so that different information can be transferred at different times over a
22 small number of communication lines. Moreover, since the two processors operate
23 in lockstep, there is no need to provide bus arbitration, framing, or other protocol
24 overhead to effect the communication between the microcontroller 232 and the
25 virtual microcontroller 220. This interface is used for, among other things,
26 transferring of I/O data from the microcontroller 232 to the FPGA 220 (since the
27 FPGA emulates only the core processor functions of the microcontroller in the
28 current embodiment). A first interface line (Data1) is a data line used by the
29 microcontroller 232 to send I/O data to the FPGA based virtual microcontroller 220.
30 This line is also used to notify the FPGA 220 of pending interrupts. This Data1 line

1 is only driven by the real microcontroller 232. A second data line (Data2), which is
2 bidirectional, is used by the microcontroller 232 to send I/O data to the FPGA based
3 virtual microcontroller of base station 218. In addition, the FPGA 220 uses the
4 Data2 line to convey halt requests (i.e., to implement simple or complex
5 breakpoints) to the microcontroller 232.

6 A third interface line is a 24/48 Mhz debug system clock used to drive the
7 virtual microcontroller 220's communication state machines (the logic used within
8 the state controller to communicate with the microcontroller 232). In the current
9 embodiment, this clock always runs at 24 MHz unless the microcontroller 232's
10 internal clock is running at 24 Mhz. In this case the system clock switches to 48
11 Mhz. Of course, these exact clock speeds are not to be considered limiting, but are
12 presented as illustrative of the current exemplary embodiment. The fourth interface
13 line is the internal microcontroller clock from the microcontroller 232.

14 A fifth line can be used to provide a system reset signal to effect the
15 simultaneous startup of both microcontrollers. This fifth line provides a convenient
16 mechanism to reset the microcontrollers, but in most environments, the
17 simultaneous startup can also be effected in other ways including switching of
18 power. Sixth and Seventh lines are provided in the current interface to provide
19 power and ground for power supply.

20 The base station 218's virtual microcontroller 220 communicates with the
21 microcontroller 232 via four signal and clock lines forming a part of the four line
22 interface 226 forming a part of a seven wire connection as described below. The
23 interface signals travel over a short (e.g., one foot) of CAT5 network cable. The ICE
24 transmits break commands to the microcontroller 232 via the base station 218,
25 along with register read/write commands when the microcontroller 232 is halted.
26 The microcontroller 232 uses the interface to return register information when
27 halted, and to send I/O read, interrupt vector, and watchdog information while
28 running. The microcontroller 232 also sends a copy of its internal clocks for the
29 ICE. The four lines of the four line interface are the first four entries in the table
30 below. Each of the signals and their purpose is tabulated below in **TABLE 1**:

Signal Name	Signal Direction with Respect to Base Station 218	Description
U_HCLK (Data Clock)	In	24/48MHz data clock driven by microcontroller 232. This clock is used to drive the ICE virtual microcontroller communication state machines. This clock always runs at 24MHz, unless the U_CCLK clock is running at 24MHz — then it switches to 48MHz.
U_CCLK (microcontroller Clock)	In	The internal microcontroller 232 CPU clock.
U_D1_IRQ (Data1)	In	One of two data lines used by the microcontroller 232 to send I/O data to the ICE. This line is also used to notify the ICE of pending interrupts. This line is only driven by the microcontroller 232 (i.e., unidirectional).
U_D0_BRQ (Data2)	In/Out	One of two data lines used by the microcontroller 232 to send I/O data to the ICE. The ICE uses this line to convey halt requests and other information to the microcontroller 232. This line is used for bi-directional communication.
ICE_POD_RST	Out	Optional active high reset signal to microcontroller 232.
ICE_POD_PW_R	Out	Optional power supply to microcontroller 232.
ICE_POD_GND	Out	Optional ground wire to microcontroller 232.

TABLE 1

Synchronization between the microcontroller 232 and the virtual microcontroller 220 is achieved by virtue of their virtually identical operation. They are both started simultaneously by a power on or reset signal. They then track each other's operation continuously executing the same instructions using the same clocking signals. The system clock signal and the microcontroller clock signal are shared between the two microcontrollers (real and virtual) so that even if the microprocessor clock is changed during operation, they remain in lock-step.

1 In accordance with certain embodiments of the invention, a mechanism is
2 provided for allowing the FPGA 220 of base station 218 and the microcontroller 232
3 to stop at the same instruction in response to a breakpoint event (a break or halt).
4 The FPGA 220 has the ability monitor the microcontroller states of microcontroller
5 232 for a breakpoint event, due to its lock-step operation with microcontroller 232.
6 In the process of executing an instruction, an internal start of instruction cycle (SOI)
7 signal is generated (by both microcontrollers) that indicates that the device is about
8 to execute a next instruction. If a breakpoint signal (a halt or break signal - the
9 terms "halt" and "break" are used synonymously herein) is generated by the FPGA,
10 the execution of the microcontroller 232 can be stopped at the SOI signal point
11 before the next instruction starts.

12 Although the SOI signal is labeled as a signal indicating the start of an
13 instruction, the SOI signal is used for multiple purposes in the present
14 microcontroller. It is not required that the SOI signal actually indicate a start of
15 instruction for many purposes, merely that there be a convenient time reference on
16 which to base certain actions. For example, any reference signal that always takes
17 place prior to execution of an instruction can be used as a time reference for
18 reading a halt command. Accordingly, any such available or generated reference
19 signal can be used equivalently as a "halt read" signal without departing from the
20 present invention. That notwithstanding, the SOI signal is conveniently used in the
21 current embodiment and will be used as a basis for the explanation that follows, but
22 should not be considered limiting.

23 Logic within the FPGA 220 of base station 218 allows not only for
24 implementation of simple breakpoint events, but also for producing breakpoints as
25 a result of very complex events. By way of example, and not limitation, a
26 breakpoint can be programmed to occur when a program counter reaches 0x0030,
27 an I/O write is happening and the stack pointer is about to overflow. Other such
28 complex breakpoints can readily be programmed to assist in the process of
29 debugging. Complex breakpoints are allowed, in part, also because the virtual
30 microcontroller 220 has time to carry out complex computations and comparisons

1 after receipt of I/O data transfers from the microcontroller 232 and before the next
2 instruction commences. After the receipt of I/O data from the microcontroller 232,
3 the FPGA 220 of base station 218 has a relatively long amount of computation time
4 to determine if a breakpoint event has occurred or not. In the event a breakpoint
5 has occurred, the microcontroller 232 can be halted and the host processor 210 is
6 informed.

7 An advantage of this process is that the FPGA 220 and the microcontroller
8 232 can be stopped at the same time in response to a breakpoint event. Another
9 advantage is that complex and robust breakpoint events are allowed while still
10 maintaining breakpoint synchronization between the two devices. These
11 advantages are achieved with minimal specialized debugging logic (to send I/O
12 data over the interface) and without special bond-out circuitry being required in the
13 microcontroller device under test 232.

14 Normal operation of the current microcontroller is carried out in a cycle of
15 two distinct stages or phases as illustrated in connection with **FIGURE 3**. The
16 cycle begins with the initial startup or reset of both the microcontroller 232 and the
17 virtual microcontroller 220 at 304. Once both microcontrollers are started in
18 synchronism, the data phase 310 is entered in which serialized data is sent from
19 the microcontroller to the virtual microcontroller. At the start of this phase the
20 internal start of instruction (SOI) signal signifies the beginning of this phase will
21 commence with the next low to high transition of the system clock. In the current
22 embodiment, this data phase lasts four system clock cycles, but this is only
23 intended to be exemplary and not limiting. The SOI signal further indicates that any
24 I/O data read on the previous instruction is now latched into a register and can be
25 serialized and transmitted to the virtual microcontroller. Upon the start of the data
26 phase 310, any such I/O read data (eight bits of data in the current embodiment)
27 is serialized into two four bit nibbles that are transmitted using the Data0 and Data1
28 lines of the current interface data portion 242. One bit is transmitted per data line
29 at the clock rate of the system clock. Thus, all eight bits are transmitted in the four
30 clock cycles of the data transfer phase.

1 At the end of the four clock cycle data transfer phase in the current
2 embodiment, the control phase 318 begins. During this control phase, which in the
3 current embodiment may be as short as two microcontroller clock periods (or as
4 long as about fourteen clock periods, depending upon the number of cycles
5 required to execute an instruction), the microcontroller 232 can send interrupt
6 requests, interrupt data, and watchdog requests. Additionally, the virtual
7 microcontroller 220 can issue halt (break) commands. If a halt command is issued,
8 it is read by the microcontroller at the next SOI signal. Once the control phase
9 ends, the data transfer phase repeats. If there is no data to transfer, data1 and
10 data2 remain idle (e.g., at a logic low state). To simplify the circuitry, I/O bus data
11 are sent across the interface on every instruction, even if it is not a bus transfer.
12 Since the virtual microcontroller 220 is operating in synchronization with
13 microcontroller 232 and executing the same instructions, the emulation system
14 knows that data transferred during non I/O read transfers can be ignored.

15 **FIGURE 4** shows this operational cycle from the perspective of the virtual
16 microcontroller 220. During the data transfer phase 310, the serialized data is
17 received over Data0 and Data1. It should be noted that prior to receipt of this I/O
18 data, the microcontroller 232 has already had access to this data for several clock
19 cycles and has already taken action on the data. However, until receipt of the I/O
20 read data during the data transfer phase 310, the virtual microcontroller 220 has not
21 had access to the data. Thus, upon receipt of the I/O read data during the data
22 phase 310, the virtual microcontroller 220 begins processing the data to catch up
23 with the existing state of microcontroller 232. Moreover, once the I/O data has been
24 read, the host computer 210 or virtual microcontroller 220 may determine that a
25 complex or simple breakpoint has been reached and thus need to issue a break
26 request. Thus, the virtual microcontroller should be able to process the data quickly
27 enough to make such determinations and issue a break request prior to the next
28 SOI. Break requests are read at the internal SOI signal, which also serves as a
29 convenient reference time marker that indicates that I/O data has been read and
30 is available for transmission by the microcontroller 232 to the virtual microcontroller

1 220.

2 By operating in the manner described, any breakpoints can be guaranteed
3 to occur in a manner such that both the virtual microcontroller 220 and the
4 microcontroller 232 halt operation in an identical state. Moreover, although the
5 virtual microcontroller 220 and the microcontroller 232 operate on I/O data obtained
6 at different times, both microcontrollers are in complete synchronization by the time
7 each SOI signal occurs. Thus, the virtual microcontroller 220 and the
8 microcontroller 232 can be said to operate in lock-step with respect to a common
9 time reference of the SOI signal as well as with respect to execution of any
10 particular instruction within a set of instructions being executed by both virtual
11 microcontroller 220 and the microcontroller 232.

12 A transfer of I/O data as just described is illustrated with reference to the
13 timing diagram of **FIGURE 5**. After the microcontroller 232 completes an I/O read
14 instruction, it sends the read data back to the base station 218 to the virtual
15 microcontroller, since the virtual microcontroller 220 of the present embodiment
16 implements only the core processor functions (and not the I/O functions). The ICE
17 system can expect the incoming data stream for an I/O read to commence with the
18 first positive edge of U_HCLK (the data or system clock) when SOI signal for the
19 following instruction is at a predetermined logic level (e.g., a logic high). Thus, at
20 time T1, the SOI signal makes a transition to a logic high and one system clock
21 cycle later at time T2, the data transfer phase 310 begins. This timing allows the
22 ICE system to get the read data to the emulated accumulator of base station 218
23 before it is needed by the next instruction's execution. Note that the first SOI pulse
24 shown in **FIGURE 5** represents the first SOI following the I/O read instruction (but
25 could be any suitable reference time signal). Transfer of the data from the
26 microcontroller 232 is carried out using the two data lines (data2 and data1, shown
27 as U_D0_BRK and U_D1_IRQ) with each line carrying four bits of an eight bit word.
28 During this data transfer phase 310, an eight bit transfer representing the I/O read
29 data can take place from the microcontroller 232 to the base station 210 in the four

1 clock cycles between T2 and T3. The control phase 318 starts at time T3 and
2 continues until the beginning of the next data transfer phase 310. The SOI signal
3 at T4 indicates that the next data transfer phase is about to start and serves as a
4 reference time to read the data2 line to detect the presence of any halt signal from
5 the virtual microcontroller 220. The current control phase 318 ends at T5 and the
6 next data transfer phase 310 begins.

7 The base station 218 only transmits break (halt) commands to the
8 microcontroller 232 during the control phase. After the microcontroller 232 is halted
9 in response to the break command, the interface can be used to implement
10 memory / register read / write commands. The halt command is read at the SOI
11 signal transition (T1 or T4). The microcontroller 232 uses the interface to return
12 register information when halted, and to send I/O read, interrupt vector and
13 watchdog timer information while running.

14 To summarize, a break is handled as follows: The ICE asserts U_D0_BRQ
15 (break) to stop the microcontroller 232. When the ICE asserts the break, the
16 microcontroller 232 reads it at the SOI transition to high and stops. The ICE assert
17 breaks during the control phase. The microcontroller 232 samples the U_D0_BRQ
18 line at the rising edge of SOI (at T4) to determine if a break is to take place. After
19 halting, the ICE may issue commands over the U_D0_BRQ line to query the status
20 of various registers and memory locations of the virtual microcontroller or carry out
21 other functions.

22 In the case of an interrupt, if an interrupt request is pending for the
23 microcontroller 232, the system asserts U_D1_IRQ as an interrupt request during
24 the control phase of the microcontroller 232. Since the interrupt signal comes to
25 the virtual microcontroller 220 from the microcontroller 232 during the control
26 phase, the virtual microcontroller 220 knows the timing of the interrupt signal going
27 forward. That is, the interrupt signal is the synchronizing event rather than the SOI
28 signal. In case of an interrupt, there is no SOI, because the microcontroller 232
29 performs special interrupt processing including reading the current interrupt vector
30 from the interrupt controller. Since program instructions are not being executed

1 during the interrupt processing, there is no data / control phase. The virtual
2 microcontroller 220 expects the interrupt vector to be passed at a deterministic time
3 across the interface during this special interrupt processing and before execution
4 of instructions proceeds. Since the virtual microcontroller 220 of the current
5 embodiment does not implement an interrupt controller, interrupt vectors are read
6 from the interrupt controller upon receipt of an interrupt request over the interface.
7 The interrupt vector data is passed over the interface using the two data lines as
8 with the I/O read data, following the assertion of an internal microcontroller IVR_N
9 (active low) signal during the control phase. In the current embodiment, an
10 interrupt cycle is approximately 10 clock cycles long. Since the interrupt service
11 cycle is much longer than the time required to transfer the current interrupt vector,
12 the data is easily transferred using the two data lines, with no particular timing
13 issues.

14 If the microcontroller 232 undergoes a watchdog reset, it asserts the IRQ
15 (interrupt) and BRQ (break) lines indefinitely. The ICE detects this condition and
16 further detects that the microcontroller clock has stopped. This is enough to
17 establish that a watchdog reset has occurred. The ICE applies an external reset,
18 and notifies the ICE software in the host computer 210.

19 The present invention provides for full in-circuit emulation without need for
20 a special bond-out version of a DUT. This is accomplished using a minimal
21 amount of design embedded within the DUT itself. In the current embodiment, the
22 only functionality required of the production microcontroller itself is to provide for
23 transfer of data over one or two lines forming the data portion of the interface and
24 at least one clock (the data clock, the microcontroller clock is optional).
25 Commands for break, watchdog and interrupt functions are received over the same
26 two data lines. These provisions are simple to implement, and use minimal
27 circuitry. The two additional pinouts used for this function were readily
28 accommodated in the eight bit microcontroller of the current invention.

29 While the present embodiment is implemented using a processor that does
30 not use pipelined instructions, this is not to be considered limiting. As long as

1 adequate time is available to serialize and transmit data over the interface, the
2 present interface and break management techniques could equally well be
3 implemented in a pipelined processor.

4 Those skilled in the art will understand that although an FPGA is used in the
5 current exemplary embodiment, this is not to be limiting. An FPGA is used as a
6 convenient mechanism to implement the virtual microcontroller of the present
7 invention. However, other hardware and/or software equivalents could equally well
8 function without the limitation of being fabricated using an FPGA. Moreover, the
9 current invention has been explained in terms of providing in-circuit emulation of the
10 core processing functions of a microcontroller. However, the present invention can
11 be realized for any complex electronic device for which in-circuit emulation is
12 needed including, but not limited to, microprocessors and other complex large
13 scale integration devices without limitation.

14 Those skilled in the art will recognize that the present invention has been
15 described in terms of exemplary embodiments based upon use of a programmed
16 processor. However, the invention should not be so limited, since the present
17 invention could be implemented using hardware component equivalents such as
18 special purpose hardware and/or dedicated processors which are equivalents to
19 the invention as described and claimed. Similarly, general purpose computers,
20 microprocessor based computers, micro-controllers, optical computers, analog
21 computers, dedicated processors and/or dedicated hard wired logic may be used
22 to construct alternative equivalent embodiments of the present invention.

23 Those skilled in the art will appreciate that the program steps and associated
24 data used to implement the embodiments described above can be implemented
25 using disc storage as well as other forms of storage such as for example Read
26 Only Memory (ROM) devices, Random Access Memory (RAM) devices; optical
27 storage elements, magnetic storage elements, magneto-optical storage elements,
28 flash memory, core memory and/or other equivalent storage technologies without
29 departing from the present invention. Such alternative storage devices should be
30 considered equivalents.

1 The present invention, as described in embodiments herein, is implemented
2 using a programmed processor executing programming instructions that are
3 broadly described above in flow chart form that can be stored on any suitable
4 electronic storage medium or transmitted over any suitable electronic
5 communication medium. However, those skilled in the art will appreciate that the
6 processes described above can be implemented in any number of variations and
7 in many suitable programming languages without departing from the present
8 invention. For example, the order of certain operations carried out can often be
9 varied, additional operations can be added or operations can be deleted without
10 departing from the invention. Error trapping can be added and/or enhanced and
11 variations can be made in user interface and information presentation without
12 departing from the present invention. Such variations are contemplated and
13 considered equivalent.

14 While the invention has been described in conjunction with specific
15 embodiments, it is evident that many alternatives, modifications, permutations and
16 variations will become apparent to those skilled in the art in light of the foregoing
17 description. Accordingly, it is intended that the present invention embrace all such
18 alternatives, modifications and variations as fall within the scope of the appended
19 claims.

20 What is claimed is:
21